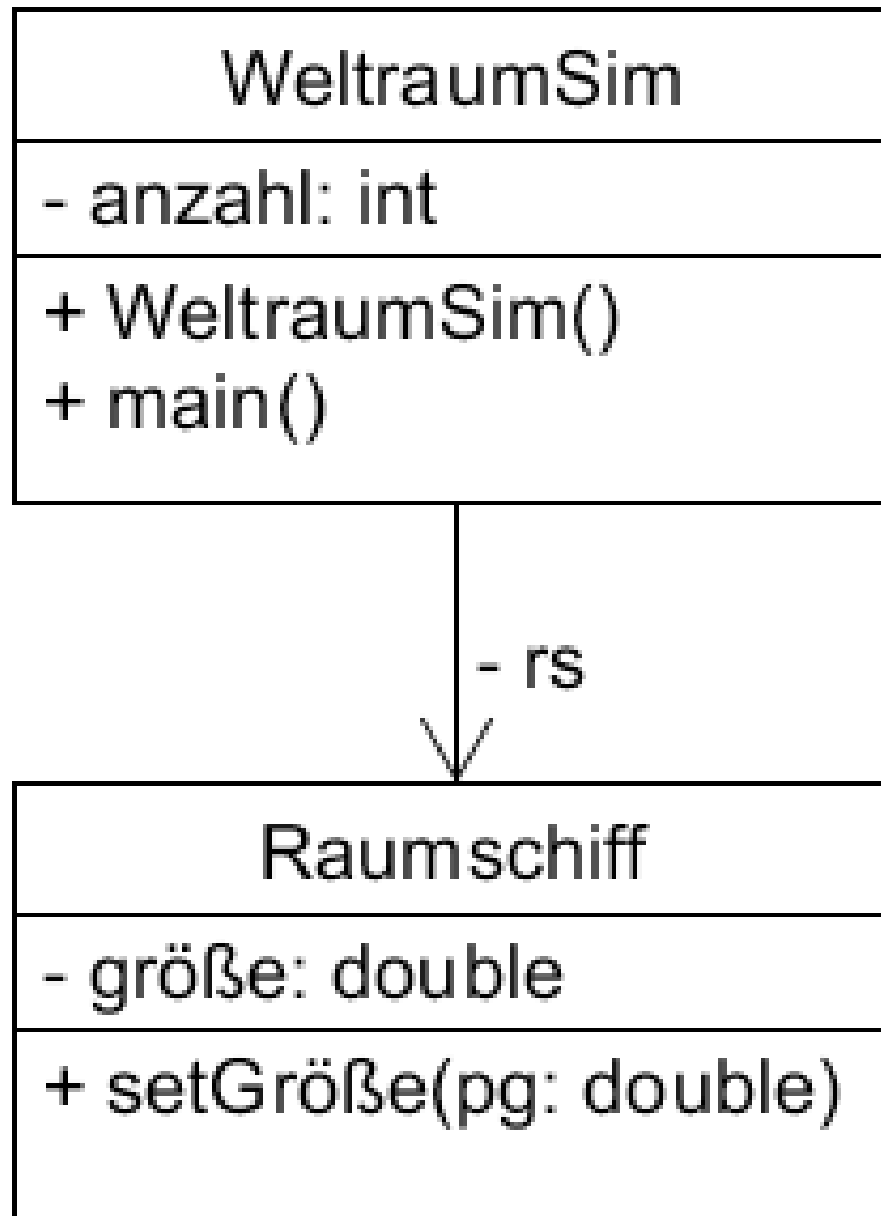


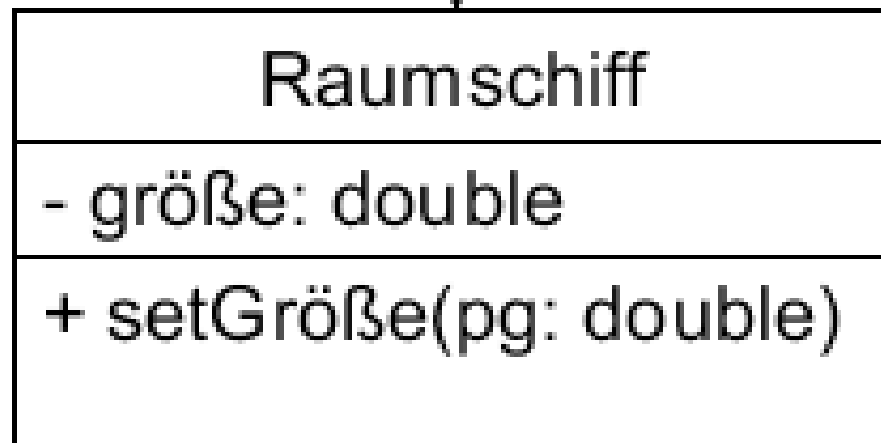
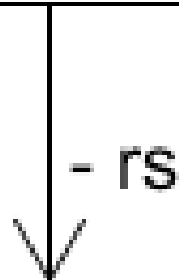
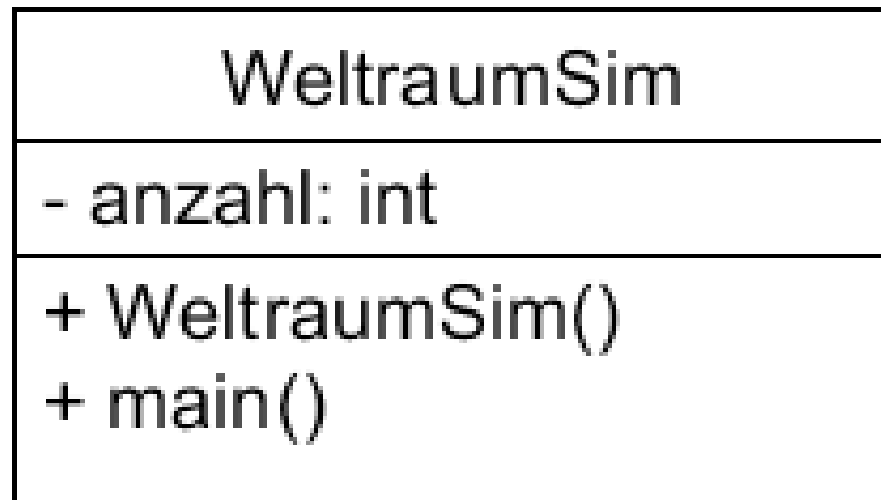
**OOP mit Java**

**Referenzen**

# Beispiel: Weltraumsimulation



# Beispiel: Weltraumsimulation



Implementierung  
WeltraumSim?

# Attribute

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;
    ...
}
```

anzahl:

Attribut mit „einfachem“ Datentyp int

rs:

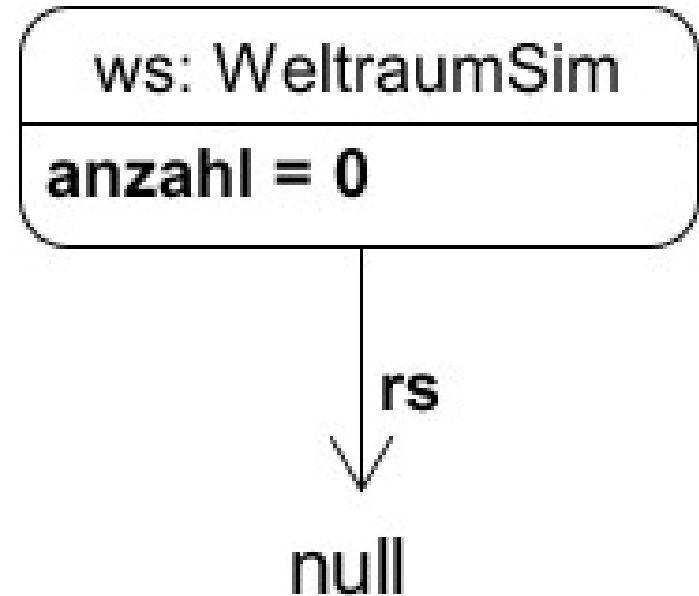
Attribut mit „komplexem“ Datentyp = Klasse

# Erinnerung: einfache Datentypen

Datentyp	Bedeutung
int	ganze Zahl
long	ganze Zahl (doppelte Bitgröße)
float	Dezimalzahl (mit Komma)
double	Dezimalzahl (doppelte Bitgröße)
char	einzelner Buchstabe
boolean	Wahrheitswert (true / false)

# Objekt der Klasse WeltraumSim:

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;
    ...
}
```



**anzahl**: Einfaches Attribut  
wird automatisch auf 0 gesetzt

**rs**: Referenz (Pfeil / Zeiger) für ein Objekt  
zeigt zunächst auf nichts („null“)

# Attribute verändern

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;

    public void main()
    {
        anzahl = 1;
        ...
    }
}
```

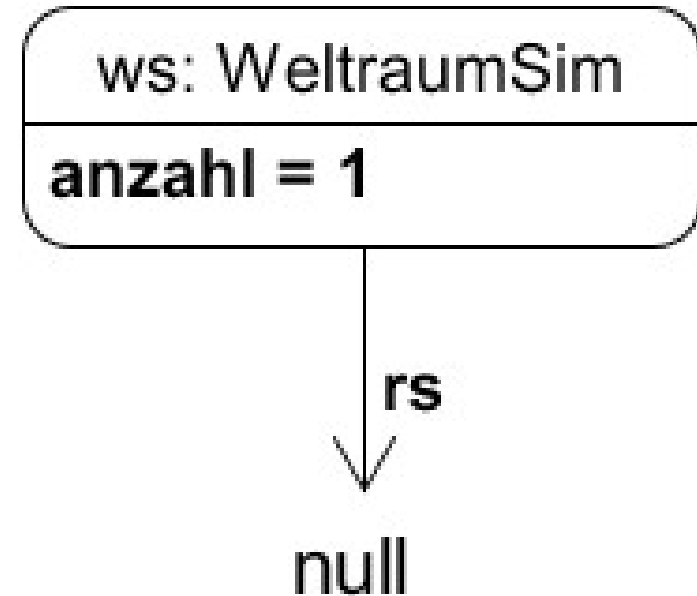
einfaches Attribut:  
Zuweisung



# Attribute verändern

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;

    public void main()
    {
        anzahl = 1;
        ...
    }
}
```





# Attribute verändern

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;

    public void main()
    {
        anzahl = 1;
        rs.setGröße(200.0);
    }
}
```

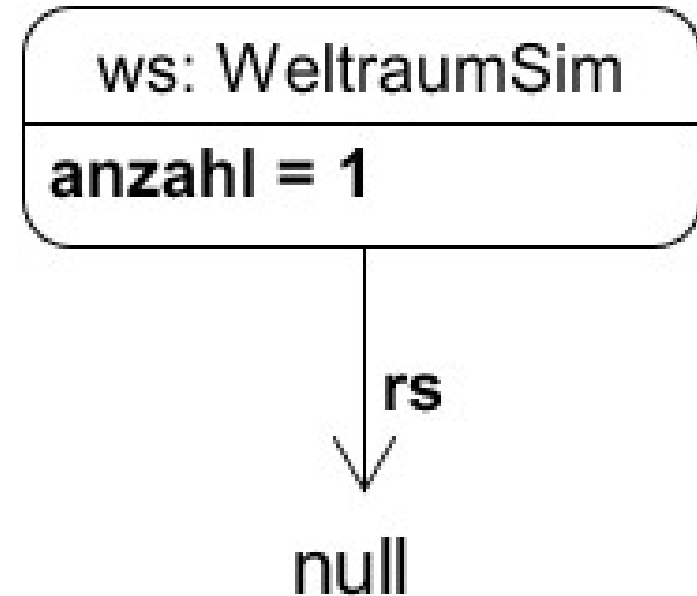
Referenz auf Objekt:  
set-Methode



# Attribute verändern

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;

    public void main()
    {
        anzahl = 1;
        rs.setGröße(200.0);
    }
}
```

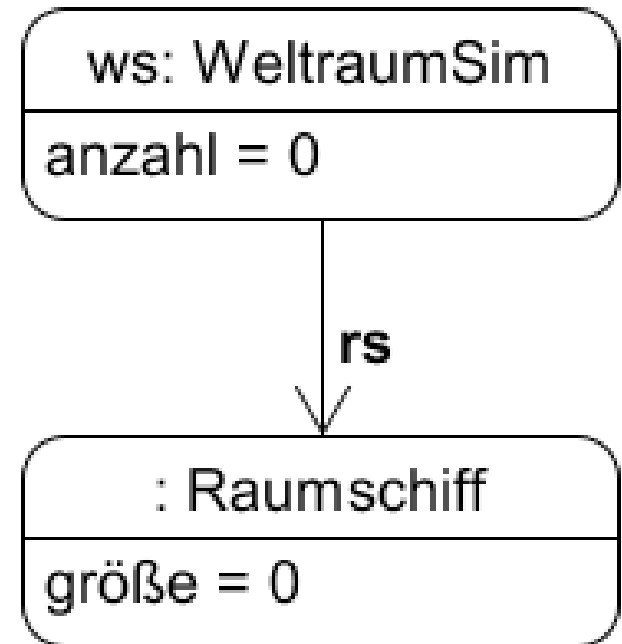


Führt zu Absturz des Programms,  
da **rs** auf kein Objekt zeigt.

# Objekt muss erzeugt werden

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;

    public WeltraumSim
    {
        rs = new Raumschiff();
    }
}
```

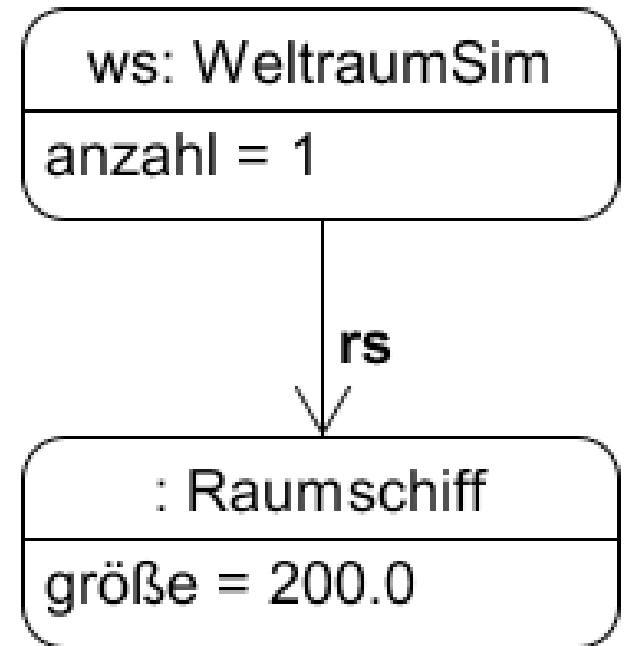


# Objekt muss erzeugt werden

```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs;

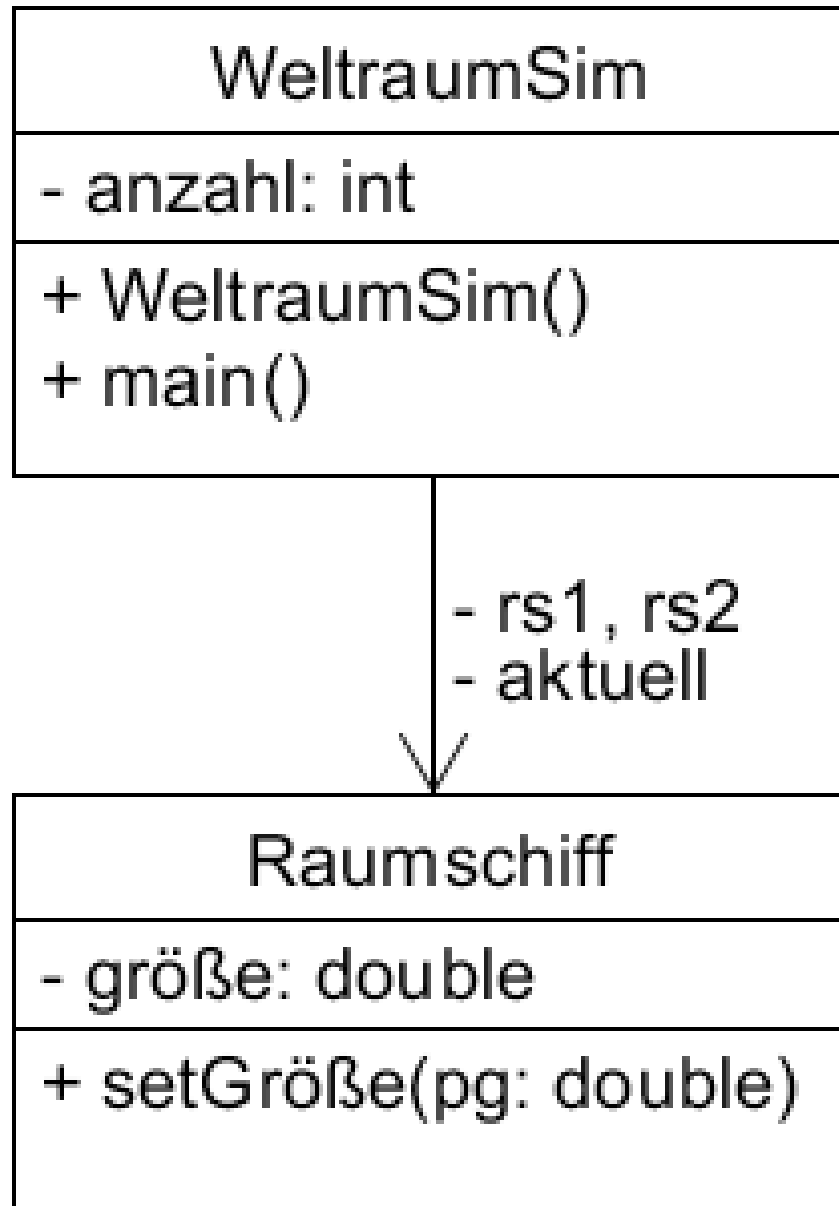
    public WeltraumSim
    {
        rs = new Raumschiff();
    }

    public void main()
    {
        anzahl = 1;
        rs.setGröße(200.0);
    }
}
```



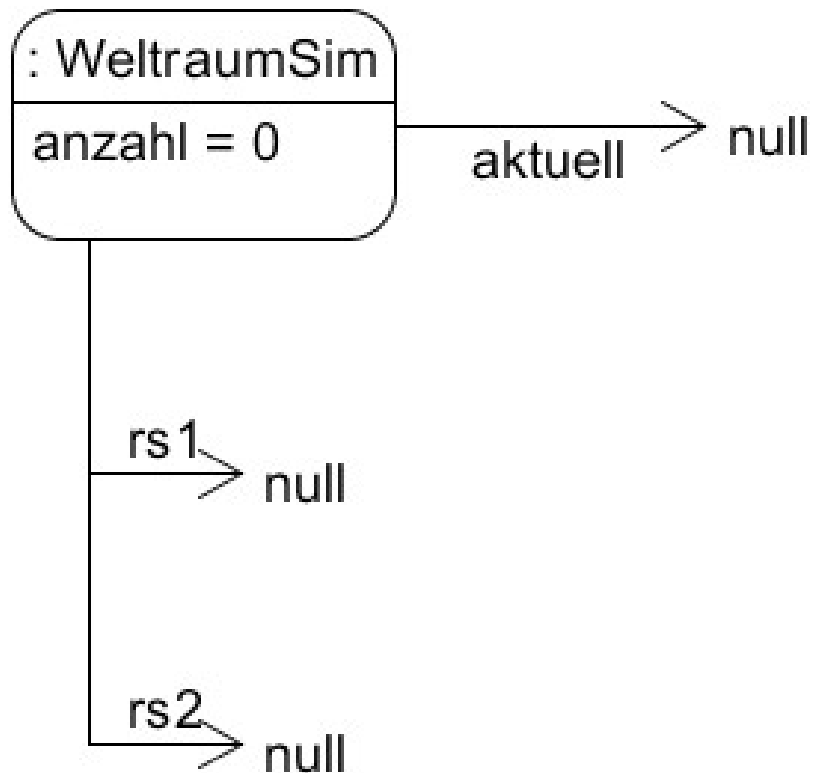
# Nutzen von Referenzen?

# Nutzen von Referenzen?



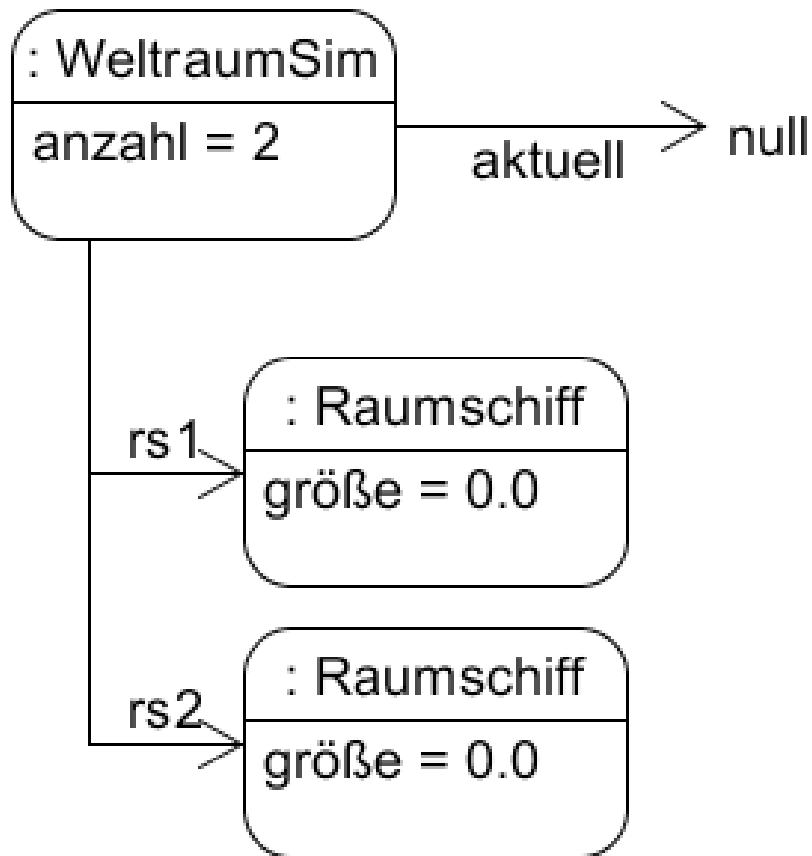
**Mehrere** Referenzen  
(Zeiger, Pfeile)  
können auf  
**das gleiche** Objekt  
zeigen.

# Beispiel



```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs1, rs2;
    private Raumschiff aktuell;
```

# Beispiel

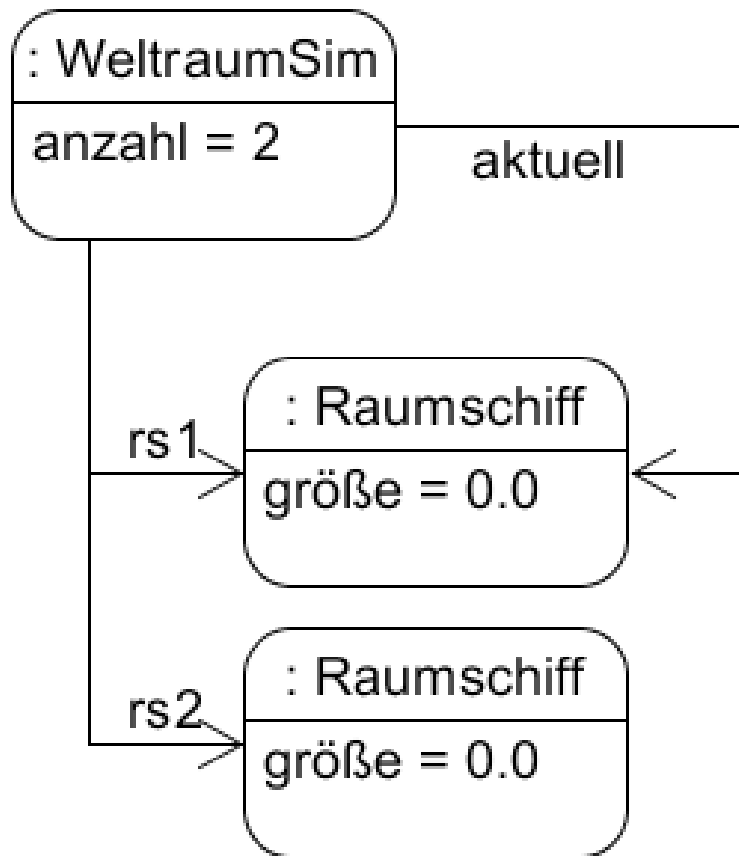


```
public class WeltraumSim
{
    private int anzahl;
    private Raumschiff rs1, rs2;
    private Raumschiff aktuell;
```

```
public WeltraumSim
{
    rs1 = new Raumschiff();
    rs2 = new Raumschiff();
    anzahl = 2;
}
```



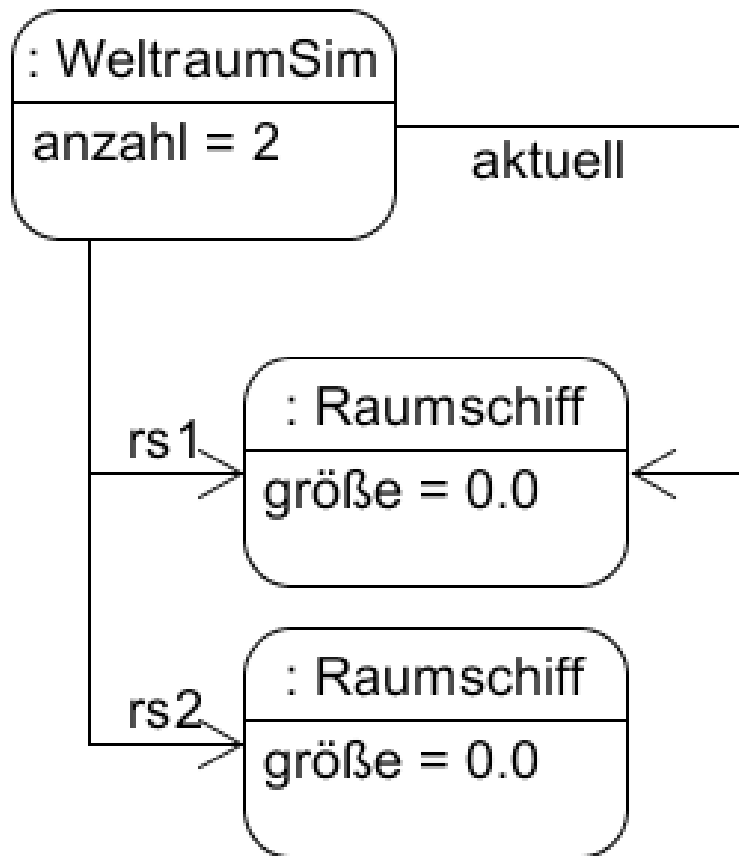
# Beispiel



...

```
public void main()
{
    aktuell = rs1;
    ...
}
```

# Beispiel



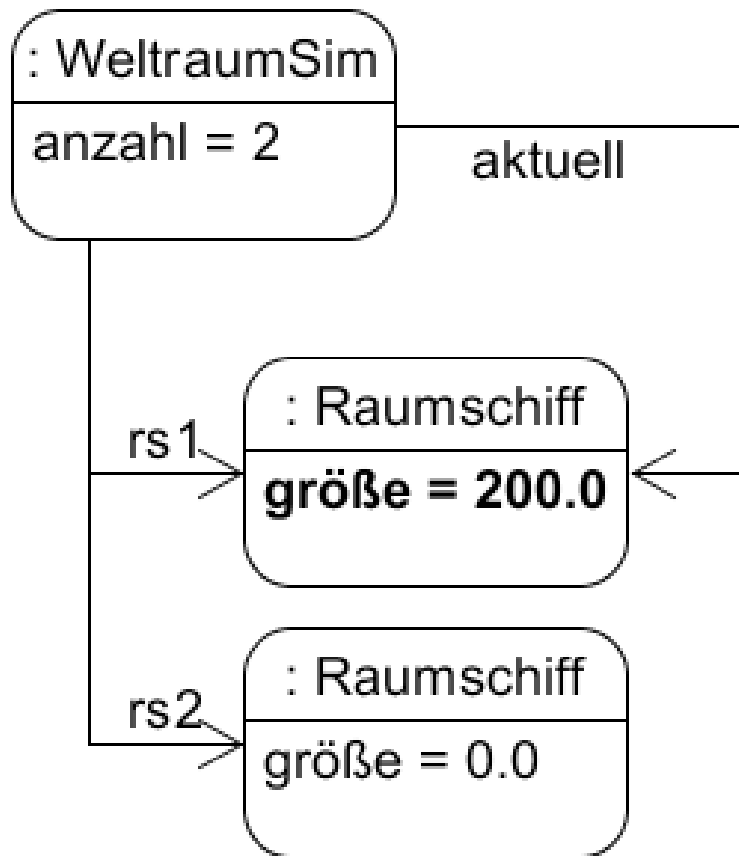
...

```
public void main()
{
    aktuell = rs1;
    ...
}
```

Zuweisung:  
Referenz aktuell zeigt auf  
das gleiche Objekt wie rs1

→ man kann das Objekt mit  
rs1 **oder** aktuell bearbeiten

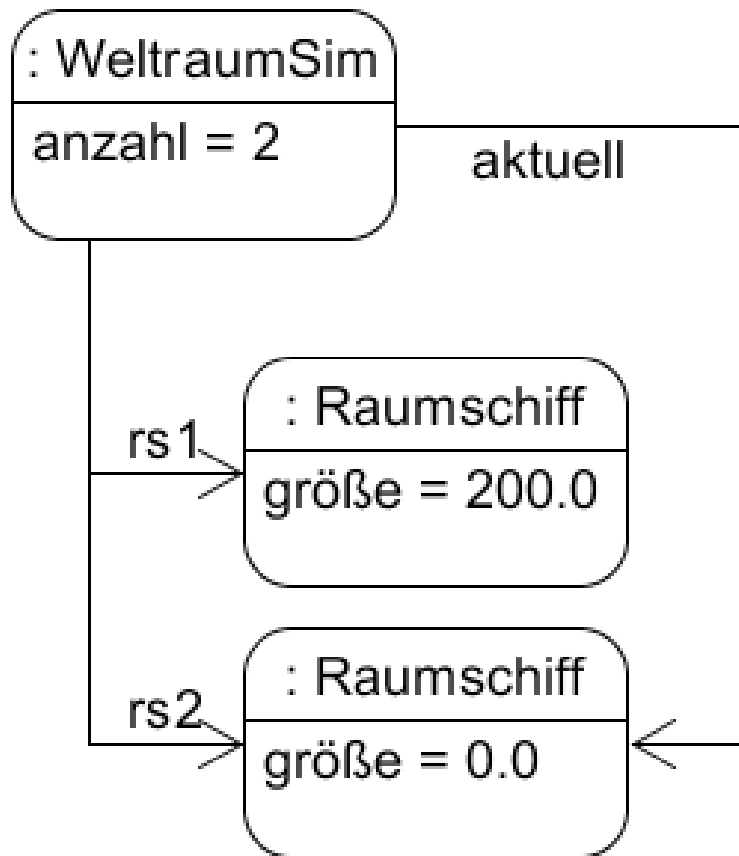
# Beispiel



...

```
public void main()
{
    aktuell = rs1;
    aktuell.setGröße(200.0);
    ...
}
```

# Beispiel

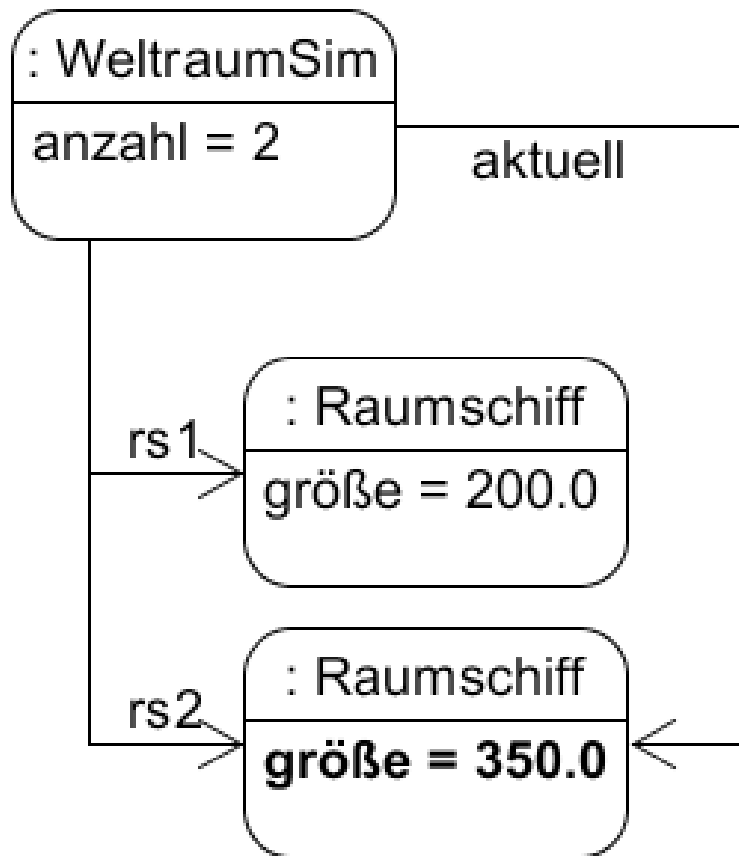


...

```
public void main()
{
    aktuell = rs1;
    aktuell.setGröße(200.0);

    aktuell = rs2;
    ...
}
```

# Beispiel



...

```
public void main()
{
    aktuell = rs1;
    aktuell.setGröße(200.0);

    aktuell = rs2;
    aktuell.setGröße(350.0);
}
```

...

# Beispiel: Benutzer kann Objekt wählen

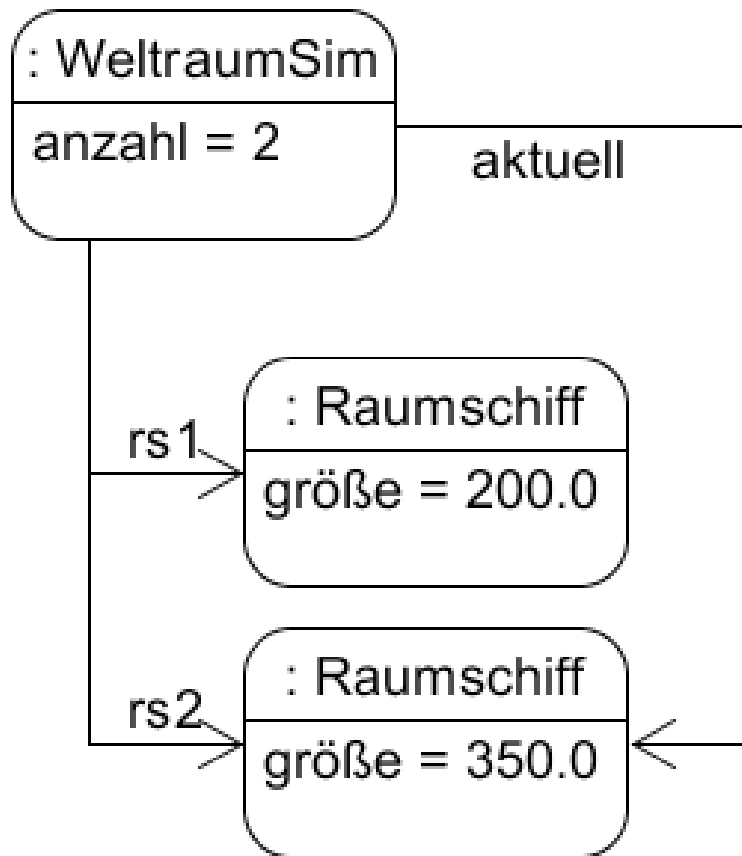
```
public void main()  
{  
    Console.println("Welches Schiff (1 / 2)?");  
    int wahl = Console.readInt();  
    if (wahl == 1)    {    aktuell = rs1;    }  
    else              {    aktuell = rs2;    }  
  
    ...  
}
```

# Weiterer Quellcode nicht festgelegt

```
public void main()
{
    Console.println("Welches Schiff (1 / 2)?");
    int wahl = Console.readInt();
    if (wahl == 1)    {    aktuell = rs1;    }
    else              {    aktuell = rs2;    }

    aktuell.ladeSchiff();
    aktuell.setzeZiel();
    aktuell.fliege();
    ...
}
```

# Löschen von Objekten



...

```
public void main()
{
```

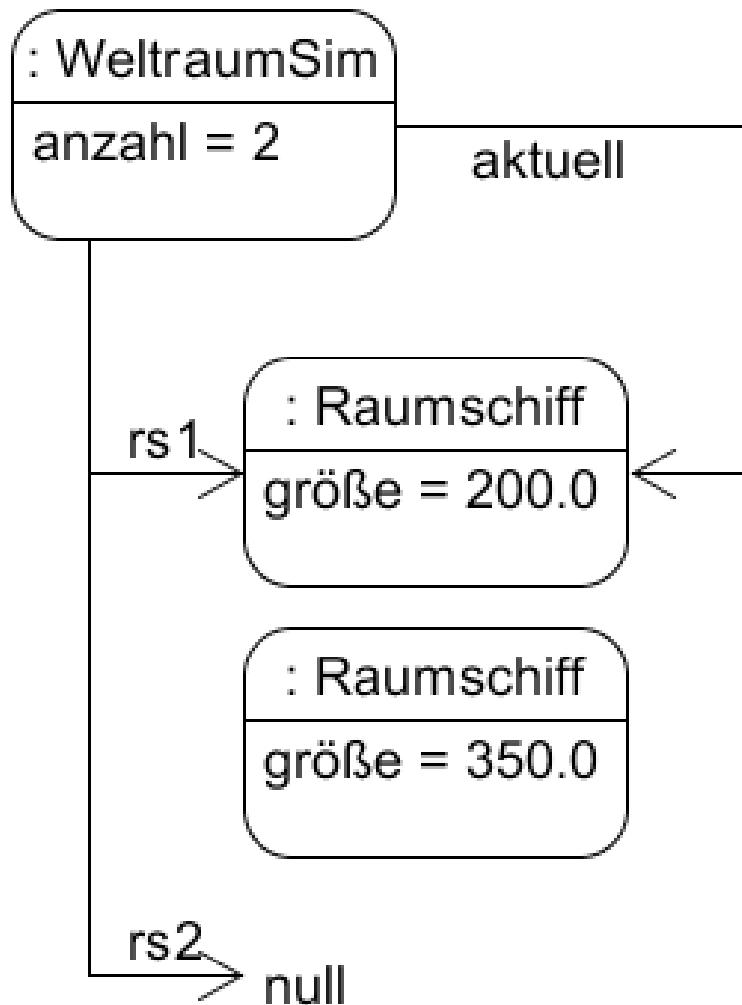
...

```
    aktuell = rs1;
    rs2 = null;
```

...



# Löschen von Objekten



...

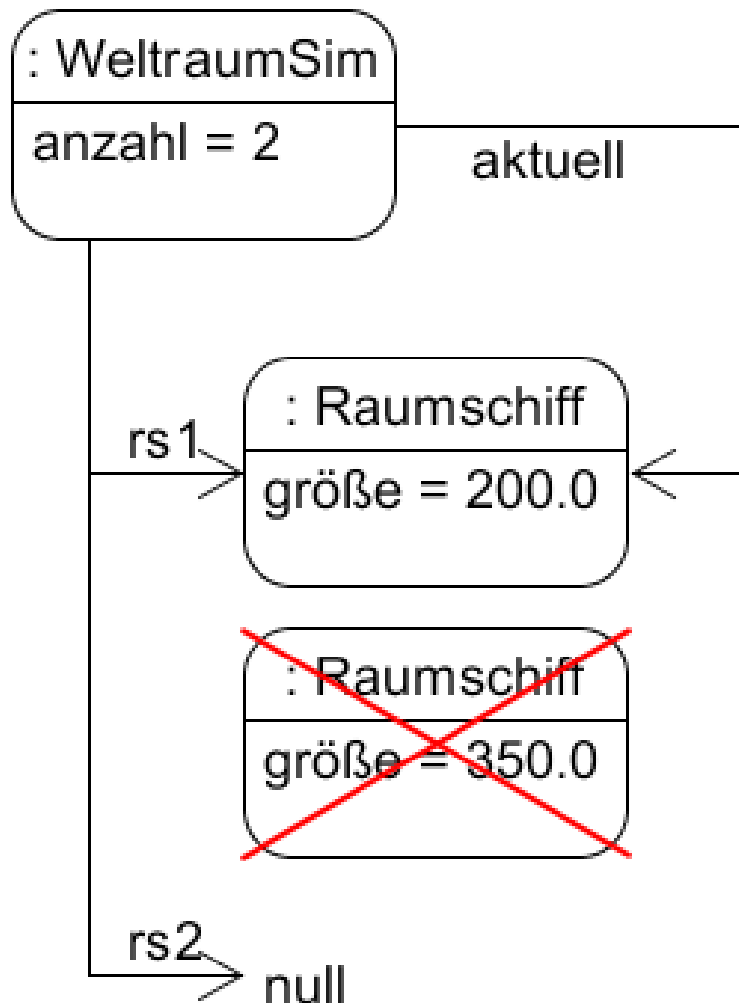
```
public void main()
{
```

...

```
    aktuell = rs1;
    rs2 = null;
```

...

# Löschen von Objekten



...

```
public void main()
{
```

...

```
    aktuell = rs1;
    rs2 = null;
```

...

Keine Referenz zeigt mehr  
auf das Objekt  
→ nicht mehr erreichbar  
→ es wird „gelöscht“

# Das Löschen von Objekten ...

... übernimmt in Java der „**Garbage Collector**“.

Er sucht permanent „im Hintergrund“ nach Objekten, auf die keine Referenzen mehr zeigen.

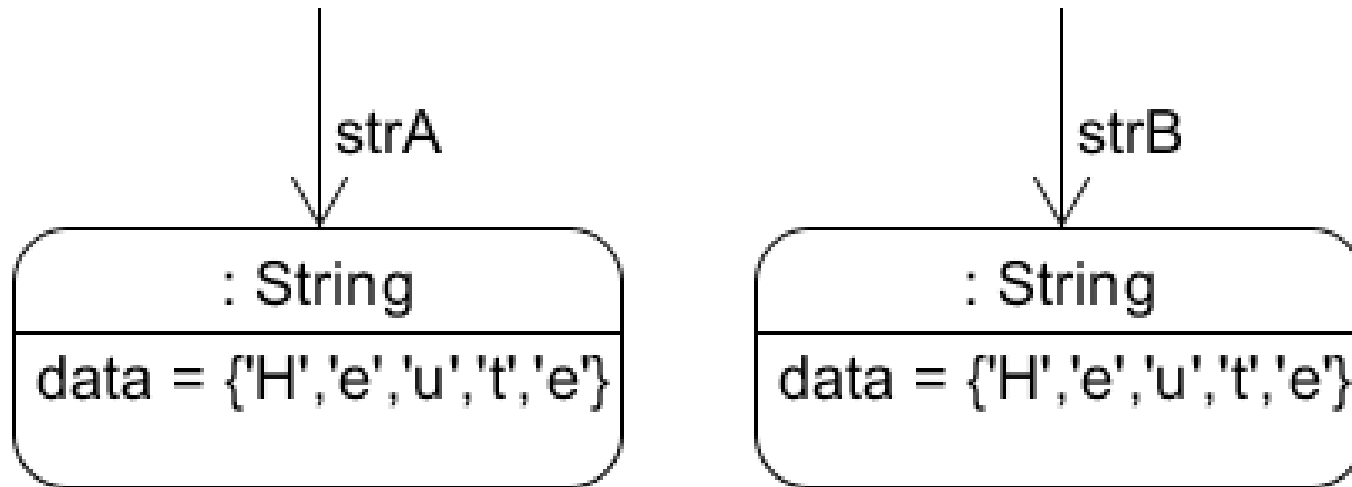
Er gibt den **Speicherplatz**, den diese Objekte benutzen, wieder frei für neue Objekte.

*Grund: früher musste man das Löschen von Objekten selbst programmieren. Das wurde oft „vergessen“, was dazu führte, dass Programme immer mehr Speicher verbrauchten  
→ „Memory Leak“.*

# Referenzen vergleichen

```
public void beispiel()  
{  
    String strA, strB;  
  
    strA = new String ("Heute");  
    strB = new String ("Heute");  
  
    if (strA == strB)  
    {    Console.println("Sind gleich."); }  
  
    else  
    {    Console.println("Sind nicht gleich."); }
```

# Referenzen vergleichen



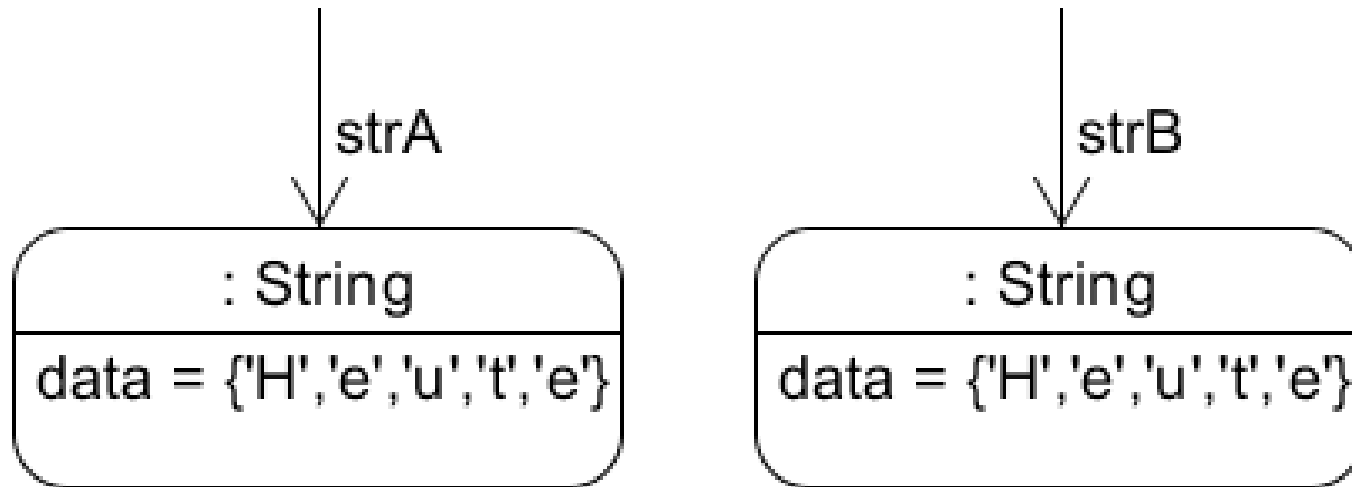
Die Objekte haben zwar den **gleichen Inhalt**,  
sind aber **verschiedene Objekte**.  
Daher sind die Referenzen **nicht gleich**.

`strA == strB` ergibt also **false**.

# Referenzen vergleichen

```
public void beispiel()  
{  
    String strA, strB;  
  
    strA = new String ("Heute");  
    strB = new String ("Heute");  
  
    if (strA.equals(strB))  
    {    Console.println("Sind gleich.");    }  
  
    else  
    {    Console.println("Sind nicht gleich.");    }
```

# Referenzen vergleichen



Um den **Inhalt** zweier Objekte zu vergleichen, verwendet man die Methode **equals()**. Sie vergleicht alle Attribute der Objekte einzeln.

`strA.equals(strB)` ergibt hier **true**.

(`strB.equals(strA)` ebenfalls)

# Zusammenfassung

- `Raumschiff rs1, rs2;`  
Deklariert zwei Referenzen (noch keine Objekte)
- `rs1 = new Raumschiff();`  
Erzeugt ein Objekt, rs1 zeigt auf dieses Objekt.
- `rs2 = rs1;`  
rs2 zeigt jetzt auf das gleiche Objekt wie rs1.
- `rs2.methode();`  
Bearbeitet das Objekt, auf das rs2 gerade zeigt.
- `rs1 == rs2`  
Vergleicht, ob rs1 und rs2 auf das gleiche Objekt zeigen
- `rs1.equals(rs2)`  
Vergleicht, ob rs1 und rs2 den gleichen Inhalt haben (auch, wenn sie auf verschiedene Objekte zeigen)



# Aufgabe

```
public void aufgabe()  
{  
    String s1, s2, s3, s4;  
    s1 = new String ("ABC");  
    s1 = new String ("XYZ");  
    s2 = new String ("ABC");  
    s3 = s1;  
    s4 = new String ("XYZ");  
}
```

a) Zeichne Objekte und Referenzen am Ende der Methode

b) Welche boolean-Werte ergeben diese Ausdrücke?

`s1 == s2`

`s1 == s3`

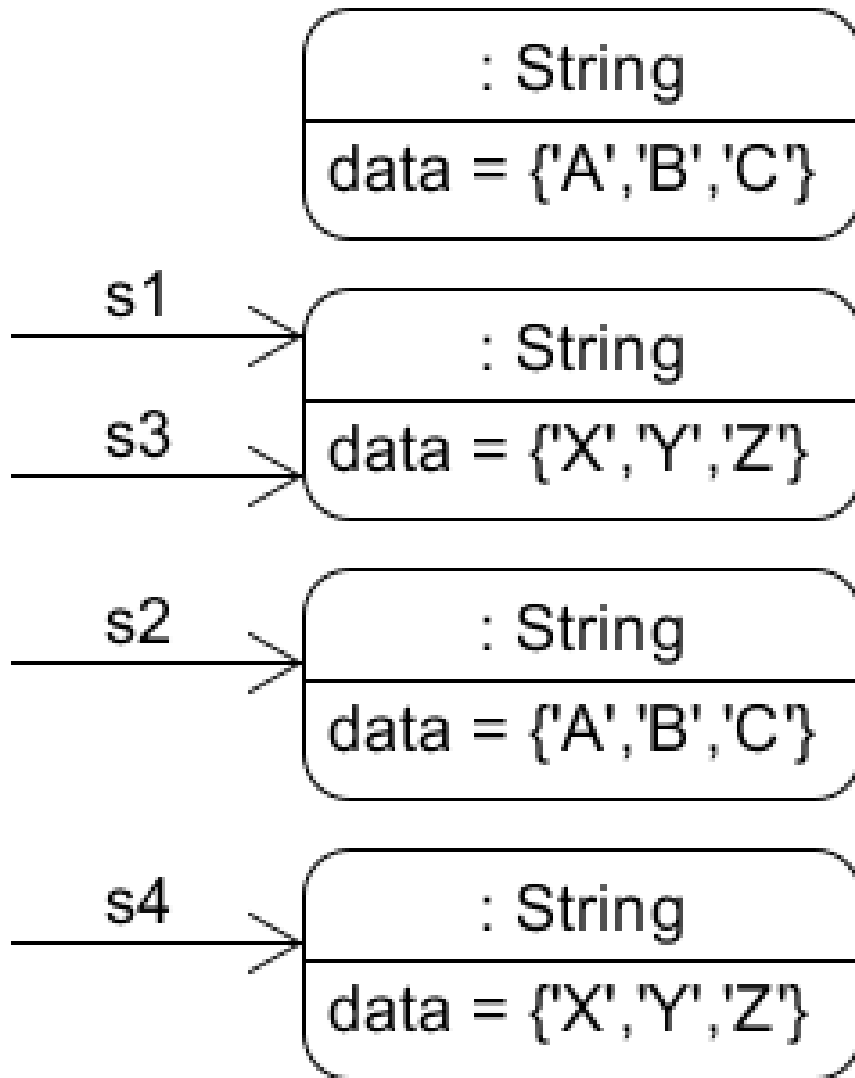
`s1 == s4`

`s1.equals(s2)`

`s1.equals(s3)`

`s1.equals(s4)`

# Lösung



`s1 == s2` → false

`s1 == s3` → true

`s1 == s4` → false

`s1.equals(s2)` → false

`s1.equals(s3)` → true

`s1.equals(s4)` → true

# Autor / Quellen

Autor:

- Christian Pothmann (cpothmann.de)  
Freigegeben unter CC BY-NC-SA 4.0, Mai 2021

